# 五种常用基数估计算法效果实验及实践建议

作者 张洋 | 发布于 2013-08-30

算法 基数估计 大数据

之前我曾写过一系列关于基数估计（cardinality estimation）算法的文章，文中介绍了一些常用基数估计算法的原理。最近对常用的基数估计算法做了一些实验，这篇文章描述了实验结果，包括这些算法的估计效果及误差状况，主要通过图表展示。通过观察实验数据和可视化图表可以加强对各种基数估计算法理论分析的直观理解。

文章首先会对实验做一些说明，然后通过图表详细展示实验数据，最后会根据实验结果总结一些实践中有用的结论。

# 实验说明

## 算法选择

这次实验共选择了五种基数估计算法，分别是：

- Linear Counting[1]
- LogLog Counting[2]
- Adaptive Counting[3]
- HyperLogLog Counting[4]
- HyperLogLog++ Counting[5]

算法实现使用我所在部门（阿里巴巴商家数据部）的开源基数估计算法库ccard-lib。

## 数据准备

哈希函数采用murmurhash32（HyperLogLog++采用murmurhash64）。

因实验结果的可靠性仅与哈希值的分布均匀性有关，而根据之前相关研究murmurhash对于顺序型数据具有良好的均匀性。因此为了简化实验，原始数据使用1-1,000,000无符号64bit整型的小端序表示。

下面将通过实验验证原始数据哈希后的均匀性。

## 实验过程

- 将原始数据经过murmurhash处理后，验证分桶数在$2^{10}$，$2^{12}$和$2^{16}$下数据的均匀性，即看各个桶的元素数量是否大致相等；同时验证各个桶中元素二进制表示的最长0前缀是否服从幂率分布。
- 对五种基数估计算法，分布记录$2^{10}$，$2^{12}$和$2^{16}$三种分桶数量下从1到1,000,000的估计值和相对误差值。取样点为100的整倍数，因此共10,000个采样点。
- 比较在$2^{10}$，$2^{12}$和$2^{16}$三种分桶数量下五种基数估计算法的误差走势。

# 实验

## 数据均匀性

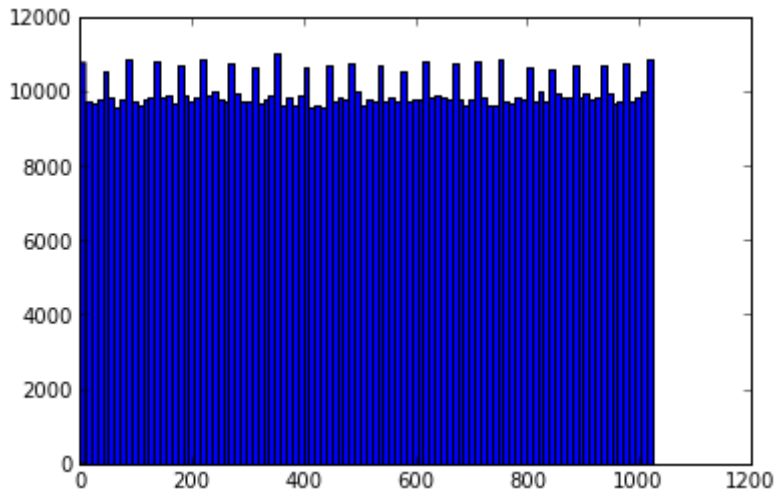
下面首先验证原始数据经过哈希后基本服从均匀分布，从而满足各种基数估计算法的基本前提条件。下面的结果通过murmurhash32哈希值给出，实际中采用murmurhash64得到了基本一致的结论。

对于32bit哈希值，分桶数为$2^p$时，用前$p$bit作为桶编号，剩下的$32-p$作为用于统计0后缀（因为均匀分布的假设，统计0后缀和0前缀是等效的，ccard-lib中除HyperLogLog++外采用统计0后缀的方式）的比特串。例如对于哈希值“01001010111010100101000000100100”，分桶数为$2^{10}$时，其桶编号为“0100101011”，即十进制的“555”，剩余部分为“10101001010000000100100”，零后缀长度为2。

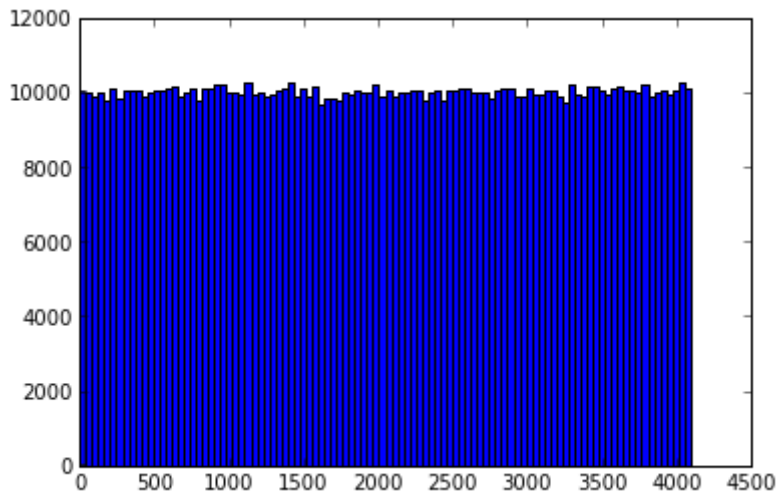
### 验证分桶均匀性

下面通过柱状图分别给出$2^{10}$，$2^{12}$和$2^{16}$三种分桶下各桶元素数量的分布，在柱状图中bins的数量均为100，因此图中每个bin并不对应一个桶。

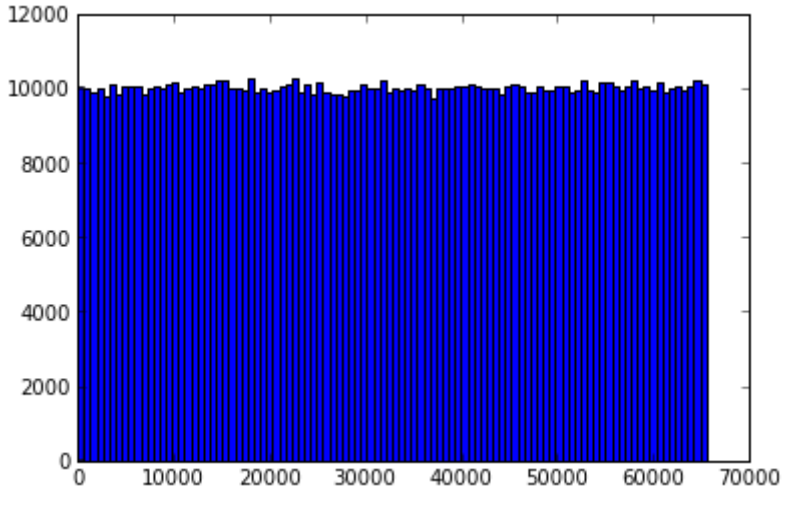
**murmurhash32哈希值分布（p=10）**




**murmurhash32哈希值分布（p=12）**




**murmurhash32哈希值分布（p=16）**



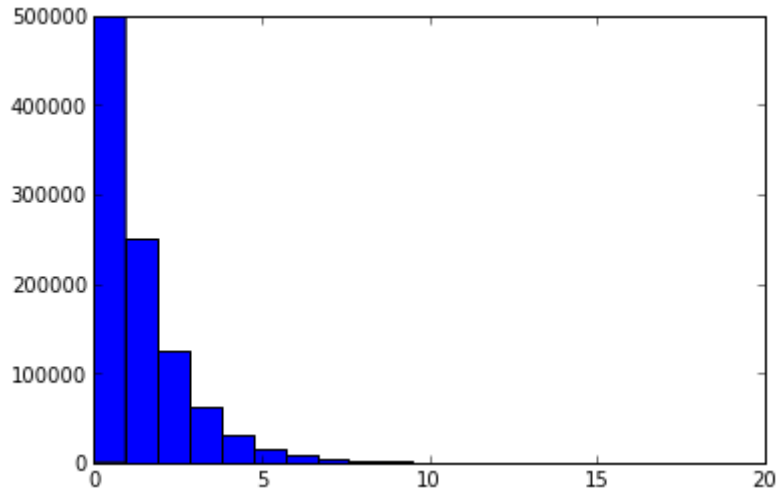

可以看到，三种分桶下数据均基本服从均匀分布。

## 0后缀长度的幂率分布性

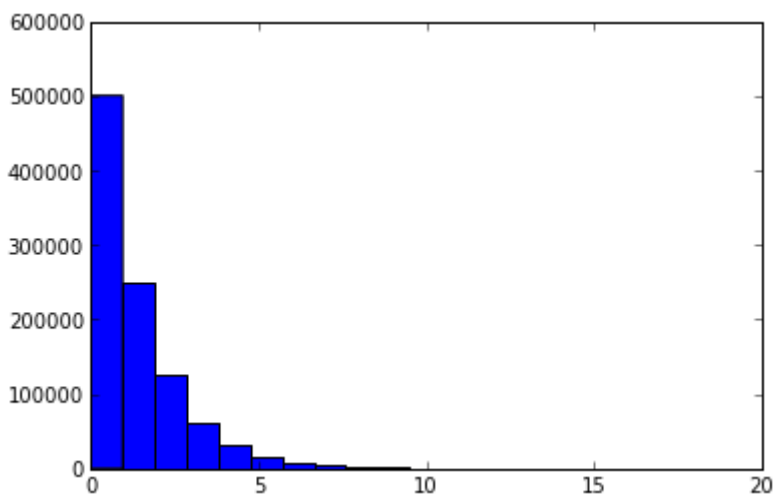
按照理论预言，如果哈希均匀性足够好，哈希剩余部分的关键统计量（最长0后缀长度）应该大约服从底数为2的幂率分布。

下图中横坐标表示0后缀长度，纵坐标表示0后缀为此长度的哈希值个数。

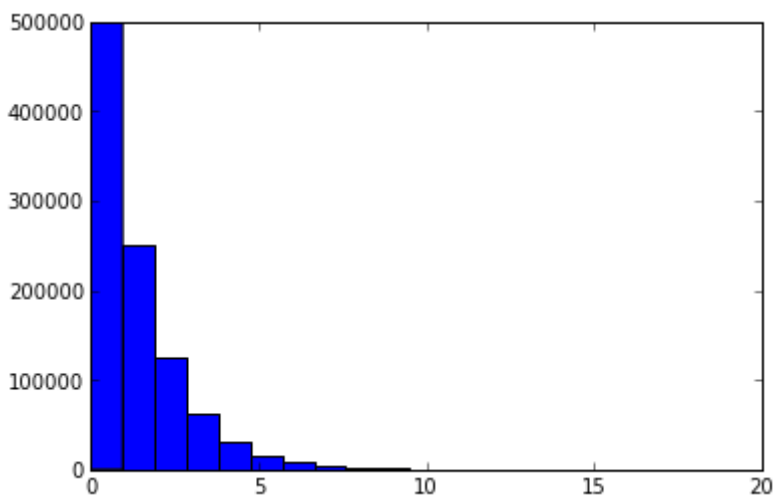
**0后缀长度分布（p=10）**




**0后缀长度分布（p=12）**




**0后缀长度分布（p=16）**




可以看到在三种分桶下统计量分布符合预期。

通过以上分析可知实验数据满足基数估计算法关于均匀性的假设。

# 基数估计算法效果

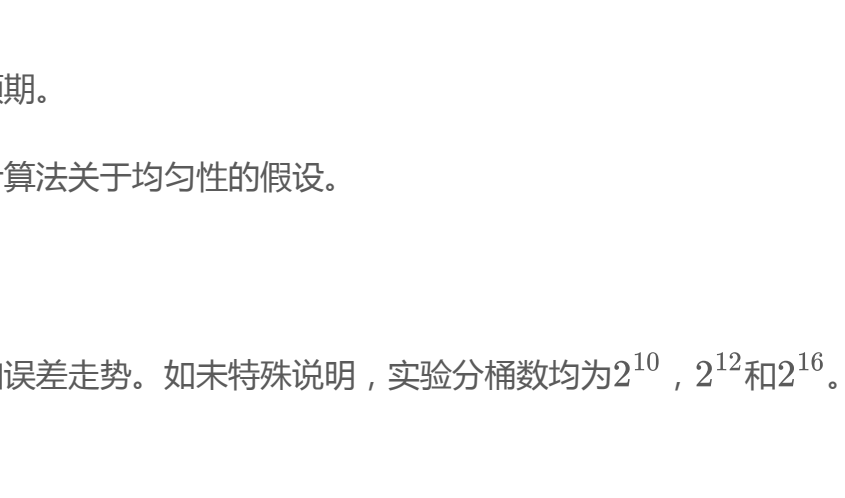
下面给出五种基数估计算法的估计效果和误差走势。如未特殊说明，实验分桶数均为$2^{10}$，$2^{12}$和$2^{16}$。

## Linear Counting

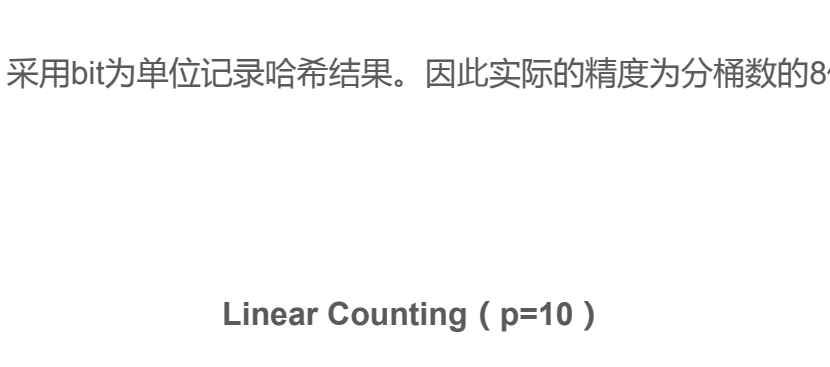
ccard-lib中当单独使用Linear Counting时，采用bit为单位记录哈希结果。因此实际的精度为分桶数的8倍，例如$2^{10}$时，实际的精度为1024*8=8192。

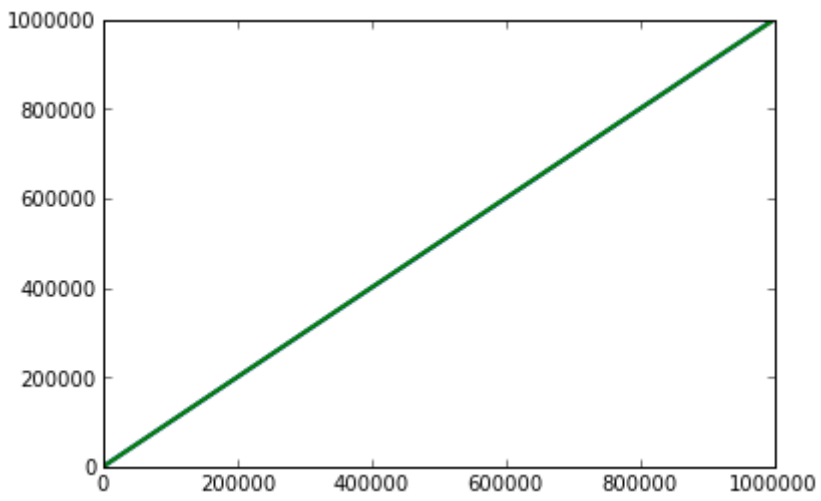
### 估计效果

**Linear Counting（p=10）**

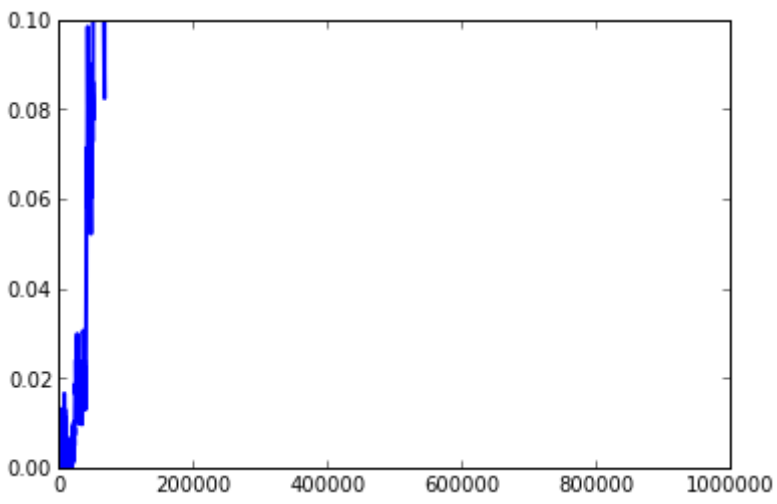


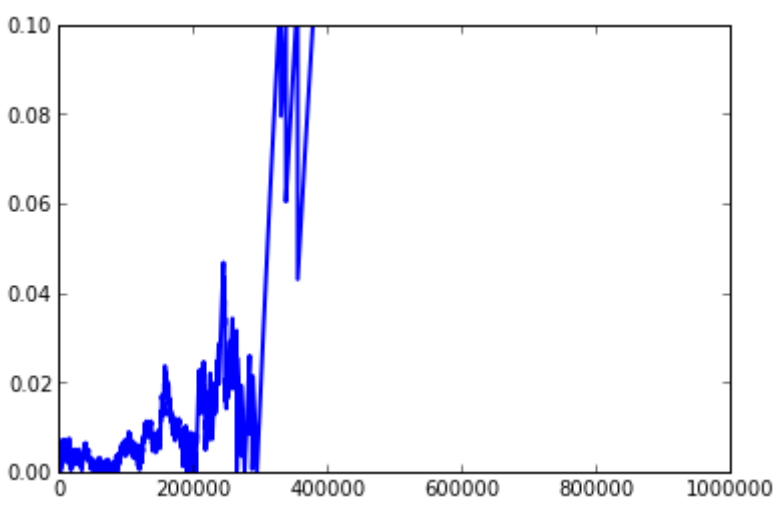

**Linear Counting ( p=12 )**

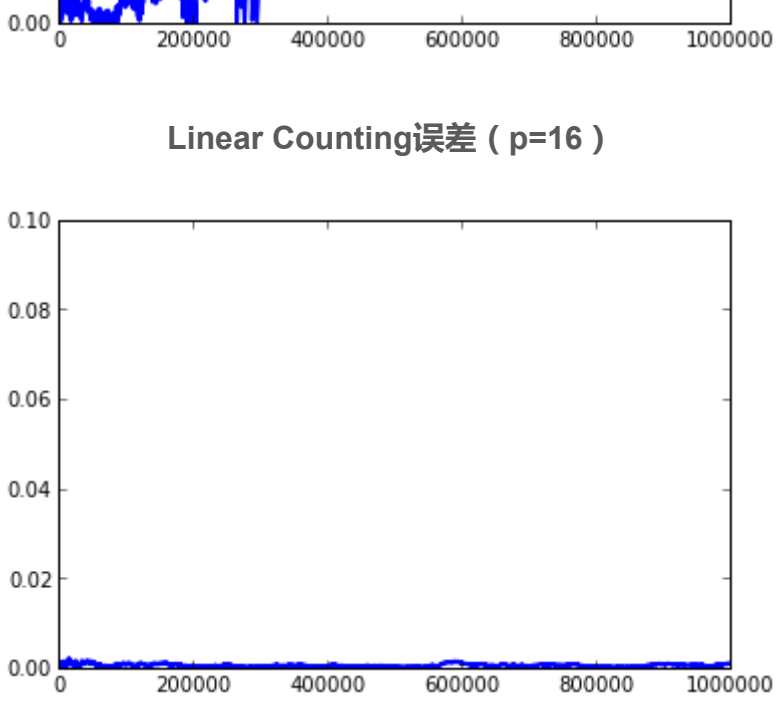





相对误差



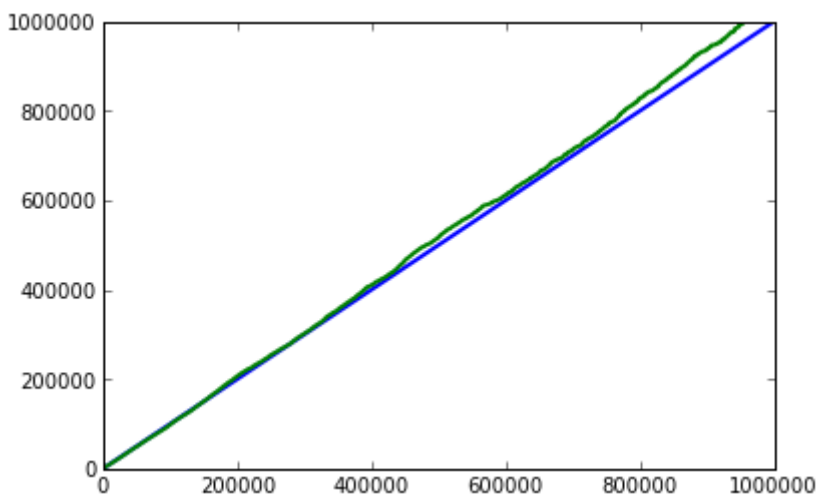




结论

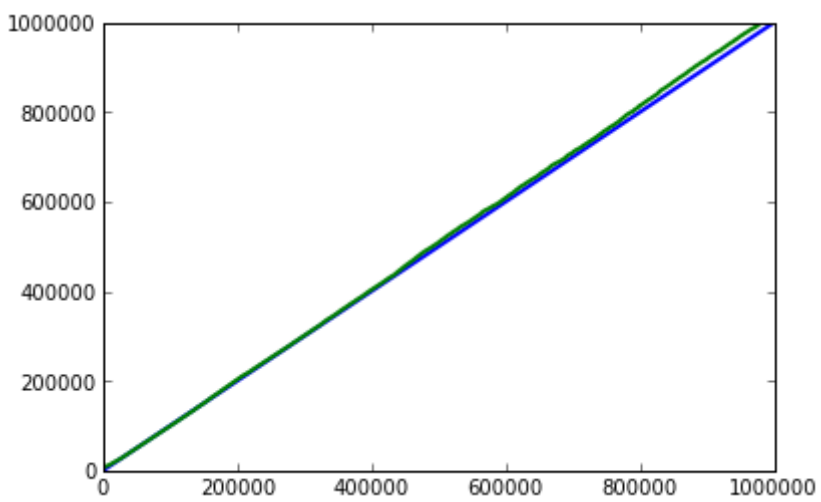
如理论预期，由于Linear Counting的有效性取决于bitmap中存在空位置，当有位置留空时，估计效果还不错，但是当bitmap全满后，Linear Counting完全失效。Linear Counting的有效估计范围线性依赖于bitmap长度。

## LogLog Counting

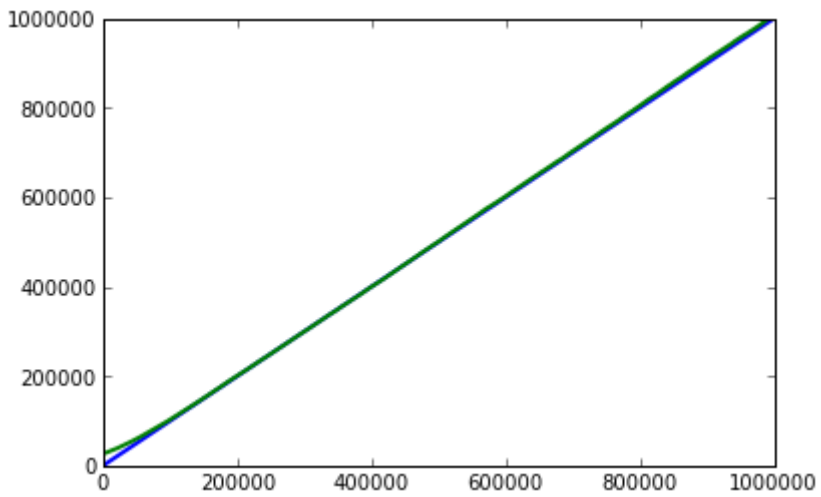
估计效果




**LogLog Counting ( p=12 )**

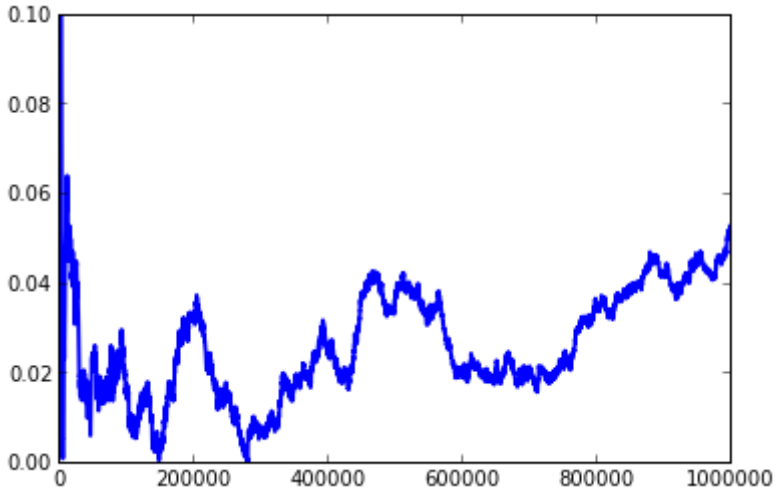



**LogLog Counting ( p=16 )**

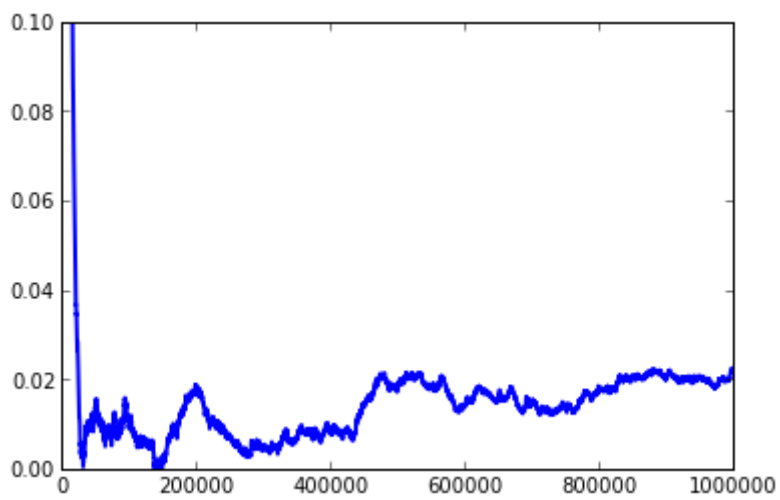



相对误差

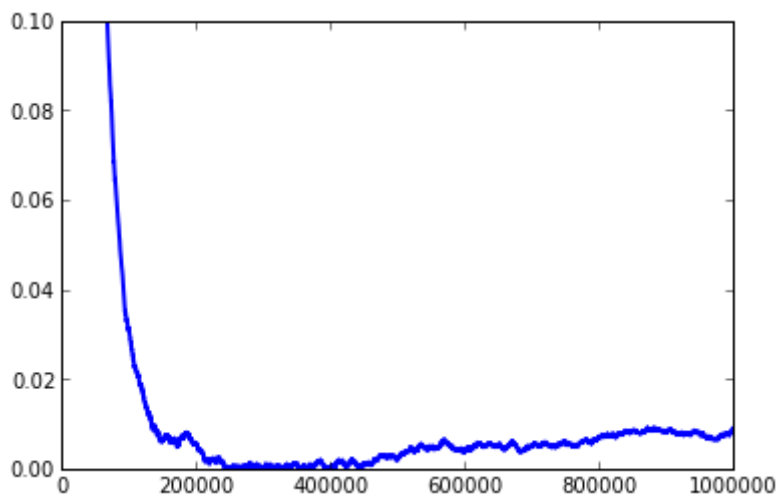
**LogLog Counting误差 ( p=10 )**




**LogLog Counting误差 ( p=12 )**



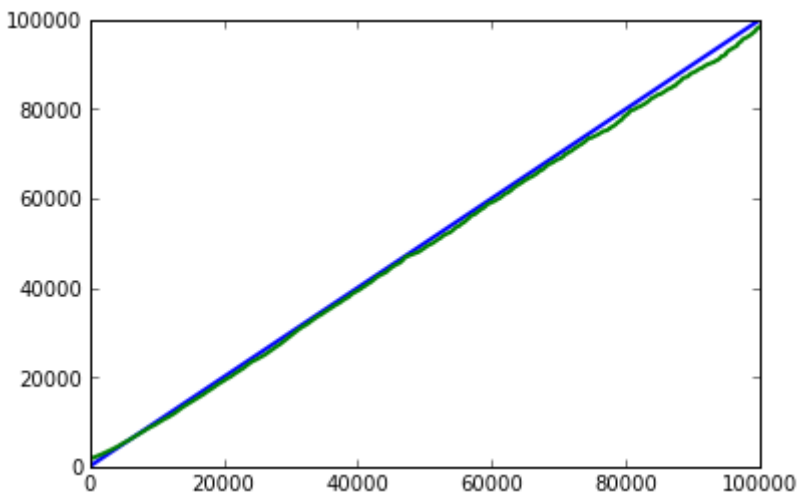

**LogLog Counting误差（p=16）**



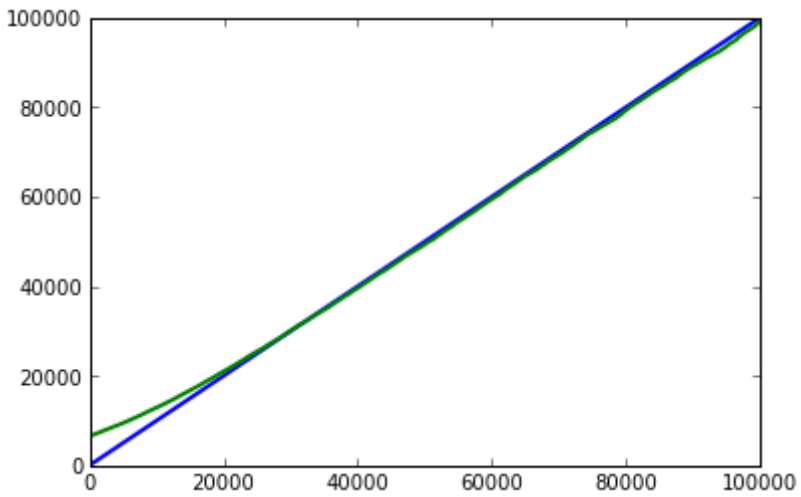

结论

LogLog Counting的表现基本与理论相符，可以看到当基数不太大的时候，LogLog Counting误差非常大，这是因为LogLog Counting在基数较小的段存在一个很大的偏差。为了明确看到这个偏差，我们截取前十分之一放大，也就是1-100,000这一段的效果图：

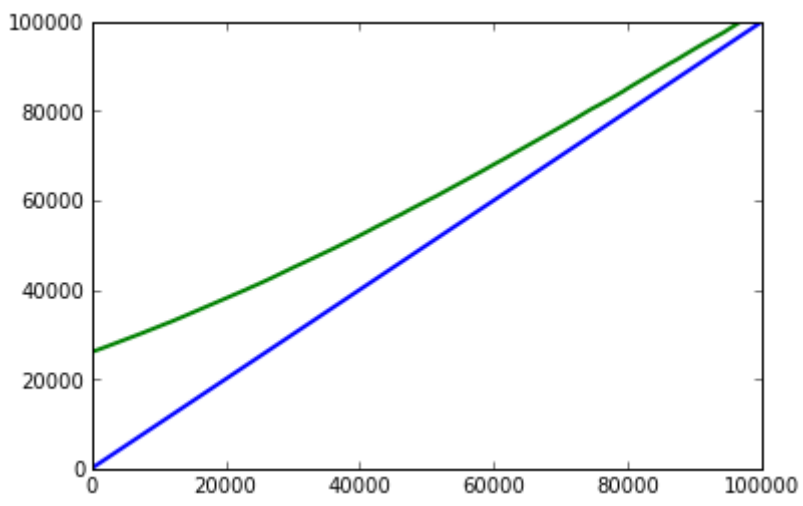
**LogLog Counting小基数区间（p=10）**




**LogLog Counting小基数区间（p=12）**



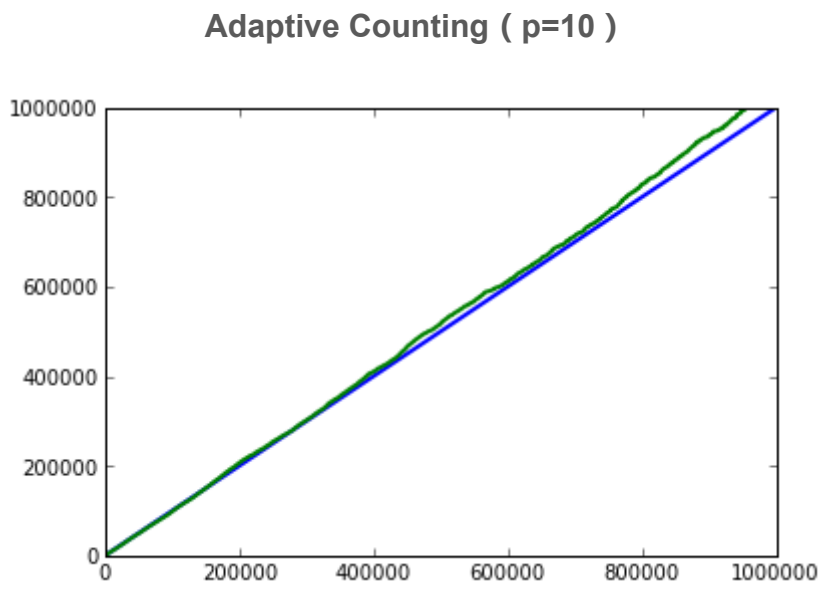

**LogLog Counting小基数区间（p=16）**

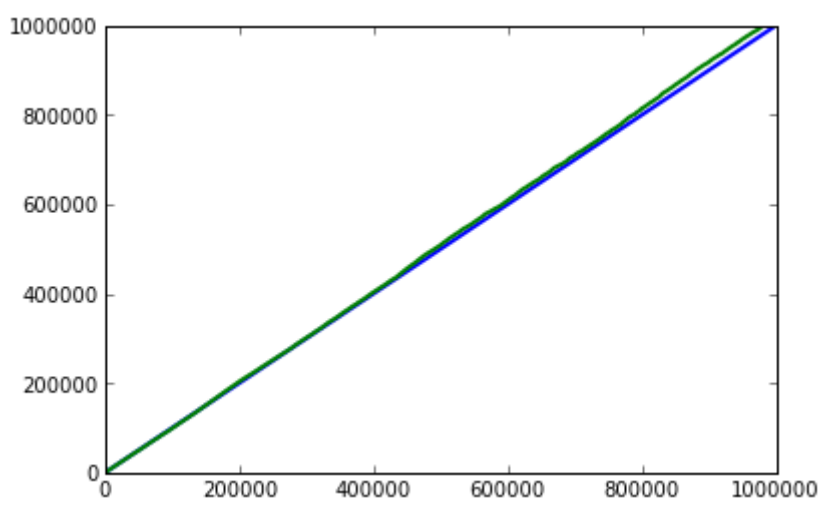


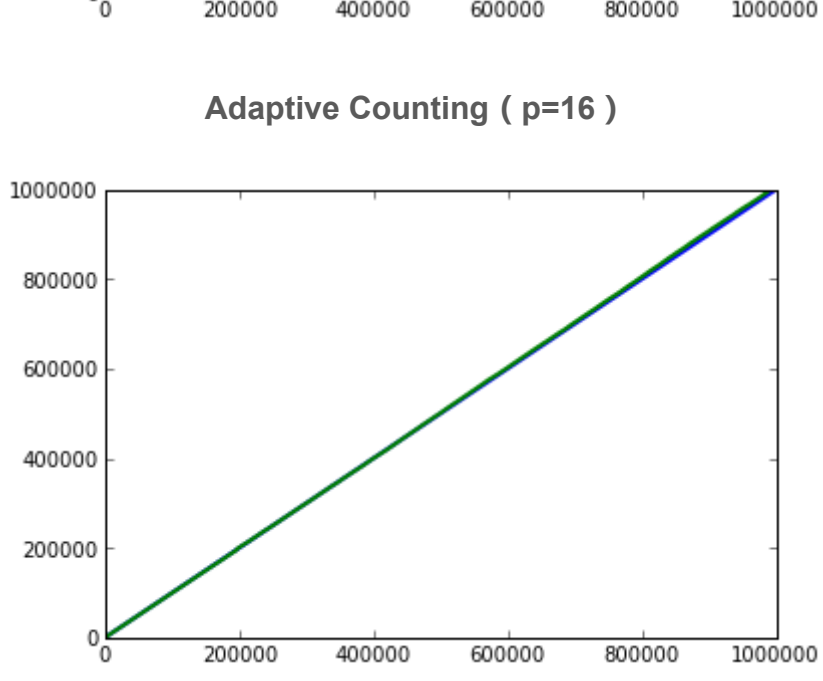

可以很明显的看到估计值严重偏离基准，而且分桶数越多这个偏差反而越明显。

## Adaptive Counting

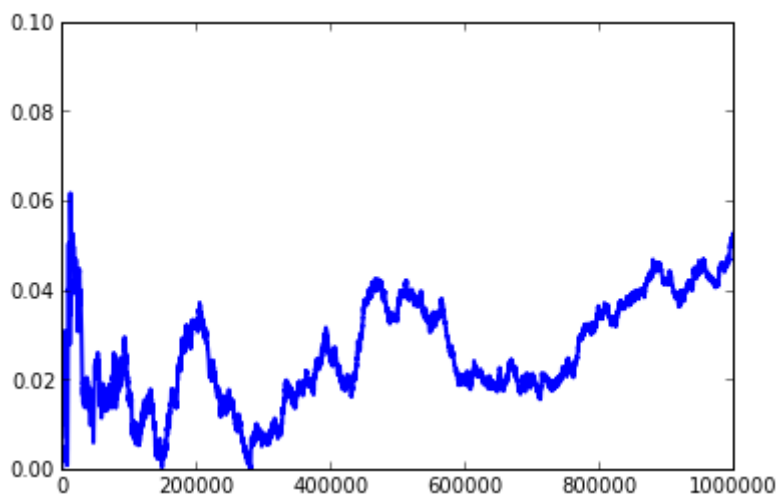
### 估计效果



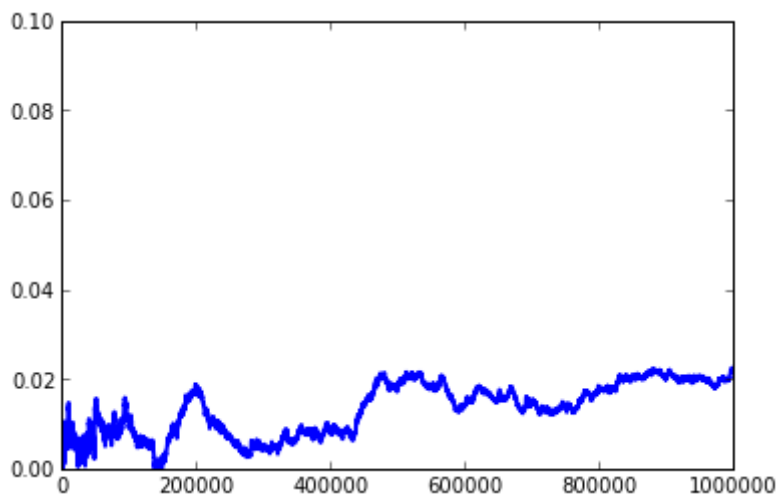



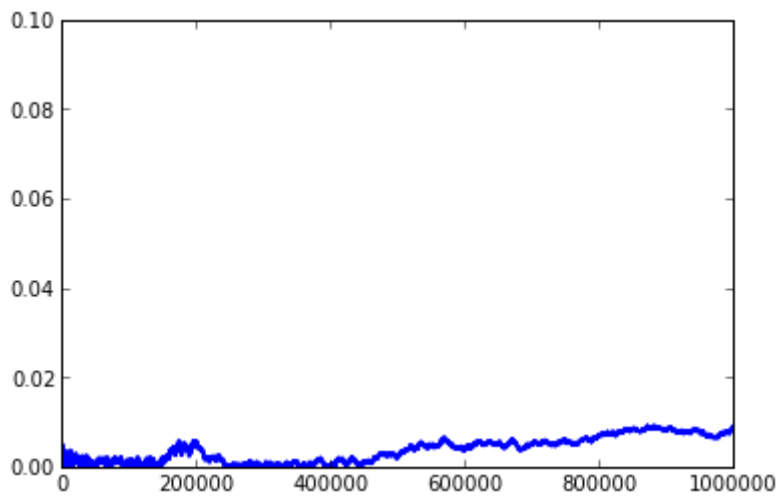

### 相对误差




**Adaptive Counting误差（p=12）**




**Adaptive Counting误差（p=16）**



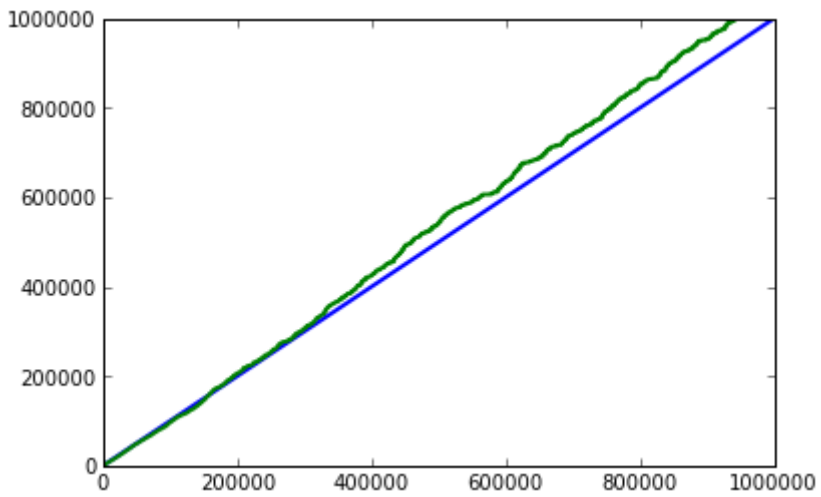

结论

由于分别在基数较小和较大时使用Linear Counting和LogLog Counting，Adaptive Counting克服了两者的缺陷，属于比较稳定的基数估计方法。而且随着分桶数的增加，估计的偏差和方差均明显减小。

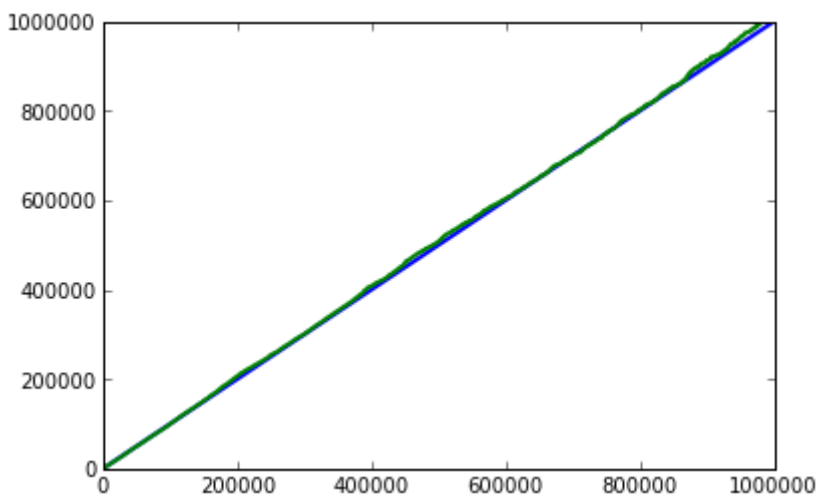
## HyperLogLog Counting

估计效果

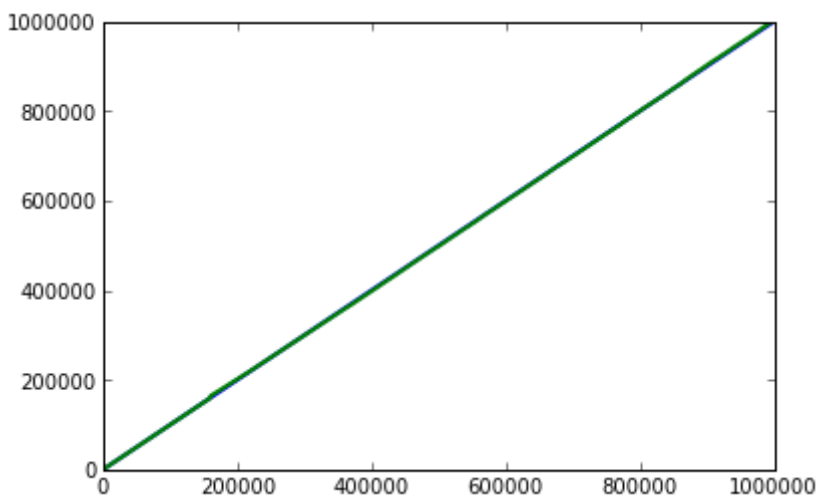
**HyperLogLog Counting（p=10）**




**HyperLogLog Counting（p=12）**

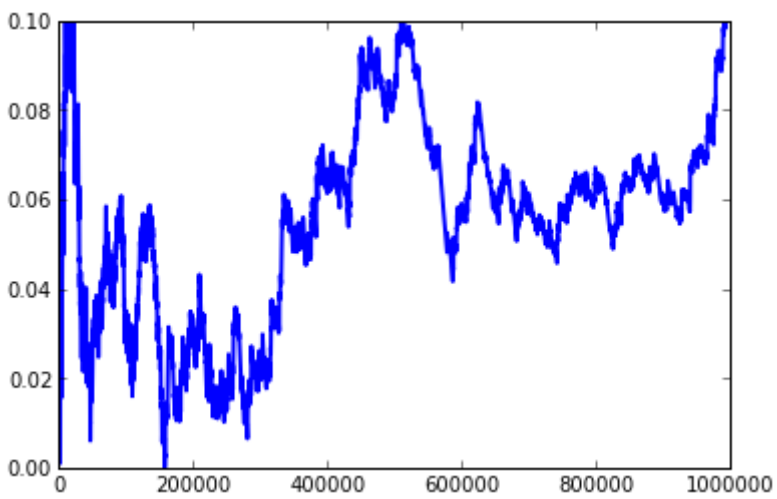



**HyperLogLog Counting ( p=16 )**

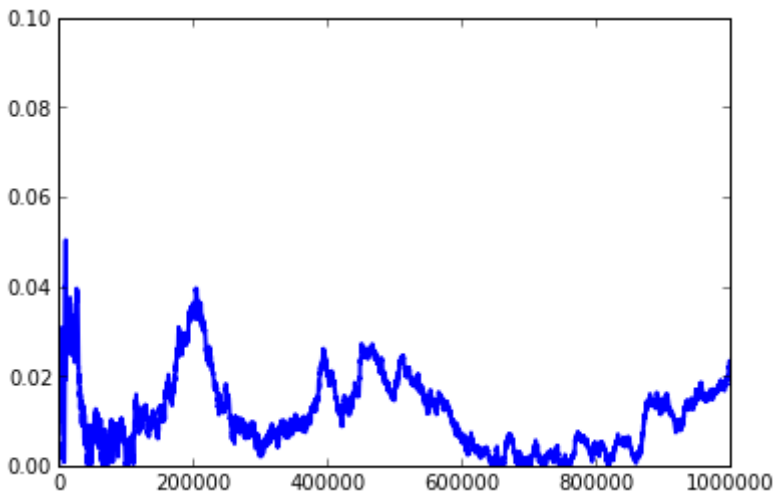



相对误差

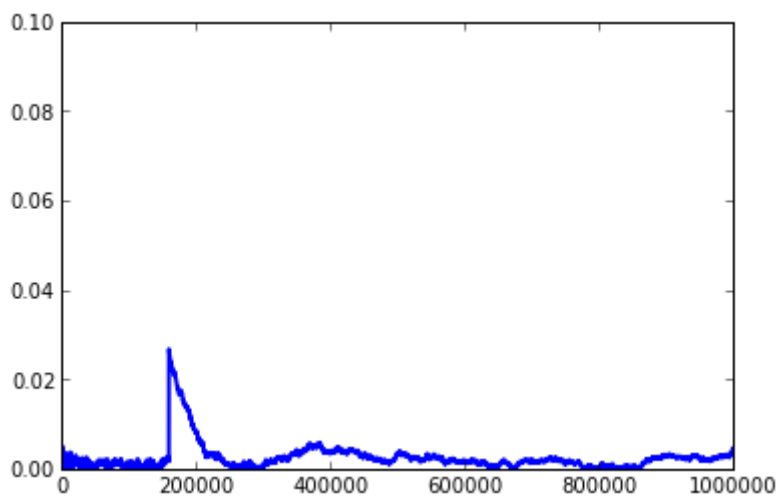
**HyperLogLog Counting误差 ( p=10 )**




**HyperLogLog Counting误差 ( p=12 )**




**HyperLogLog Counting误差 ( p=16 )**



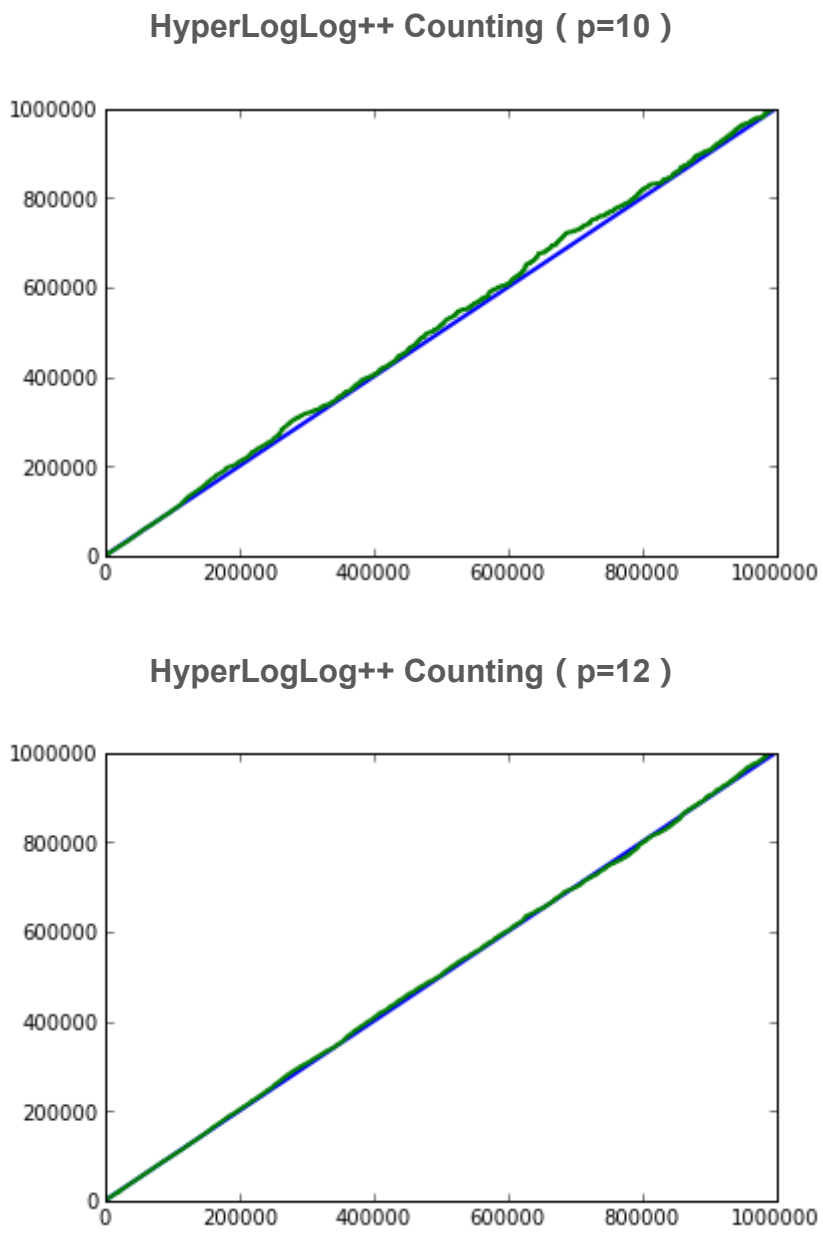

结论

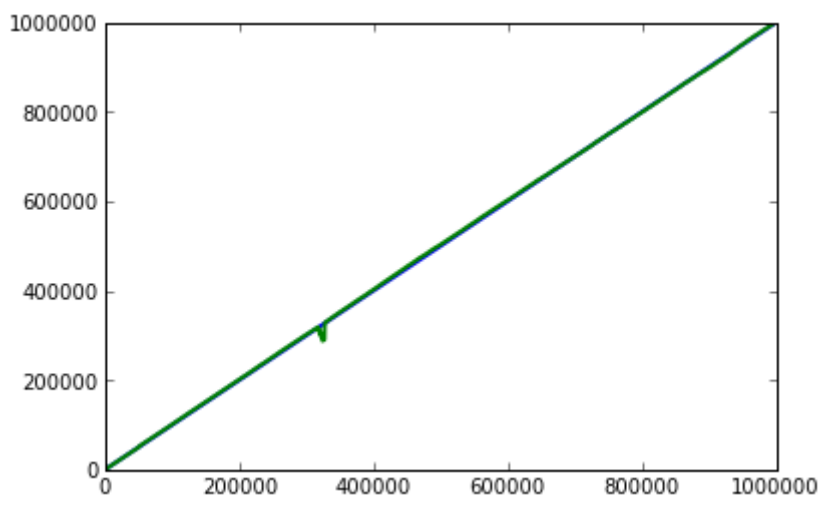
HyperLogLog Counting采用调和平均数取代LogLog Counting中的几何平均数，旨在减小离群点的影响，并且对Linear Counting转折阈值做了调整。从实验效果看，在分桶数较小时，改进效果并不明显，不过在$2^{16}$分桶下，整体偏差和稳定程度优于Adaptive Counting。

但是从误差图中可以看到，在200,000附近出现了一个明显的脉冲。其原因在Google关于HyperLogLog++ Counting的论文中[5]有分析，其主要是因为在Linear Counting刚转折后的一小段区域内存在一个偏差，HyperLogLog++ Counting的一个改进就是对这个区域的偏差进行了修正。

## HyperLogLog++ Counting

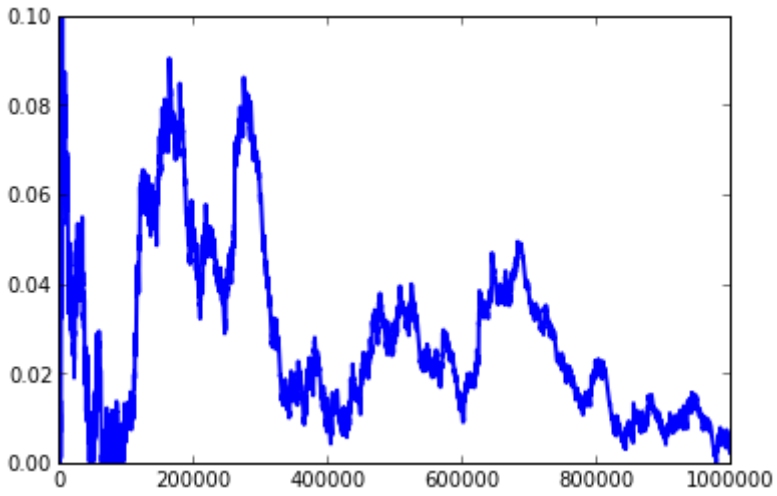
估计效果



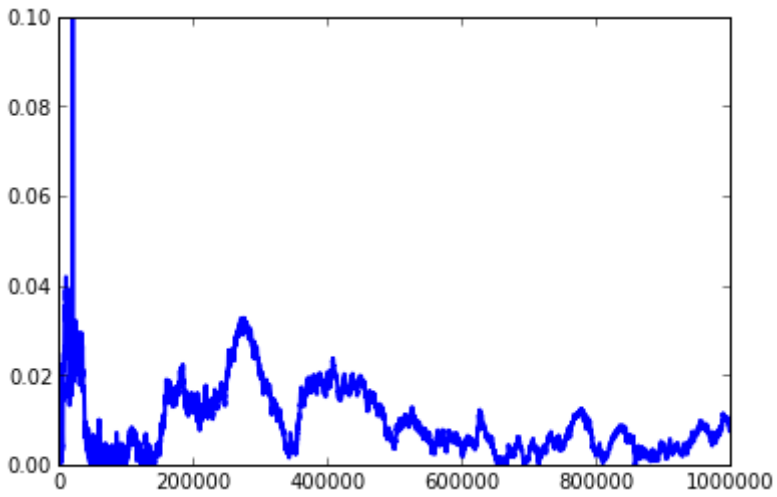




相对误差

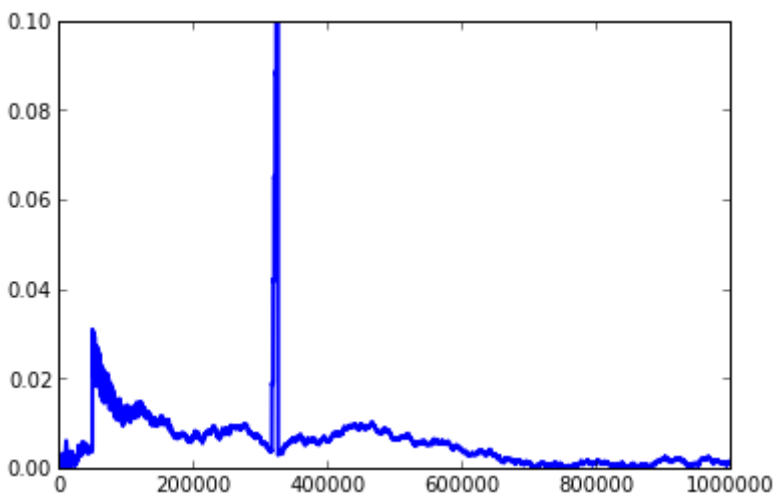
HyperLogLog++ Counting误差（p=10）




HyperLogLog++ Counting误差（p=12）




HyperLogLog++ Counting误差（p=16）




结论

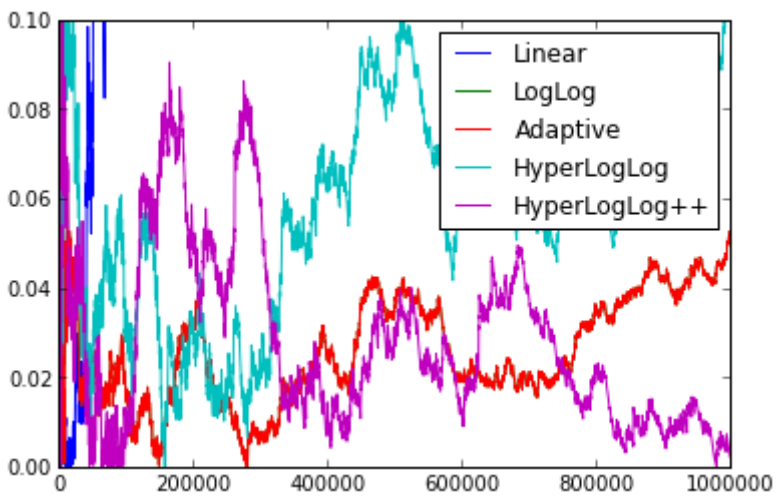
可以看到HyperLogLog++ Counting的效果非常令人失望，按论文中说法，HyperLogLog++ Counting应该比HyperLogLog Counting更准确，但实际效果不但整体偏差和方差变大，而且偏差修正的阈值明显有问题，导致一个非常明显的误差脉冲。

究其原因，个人认为HyperLogLog++ Counting中的偏差修正和转折阈值均是通过统计方法给出，并不是数学上的解析结果，因此对于不同的数据、不同的哈希可能并不通用。

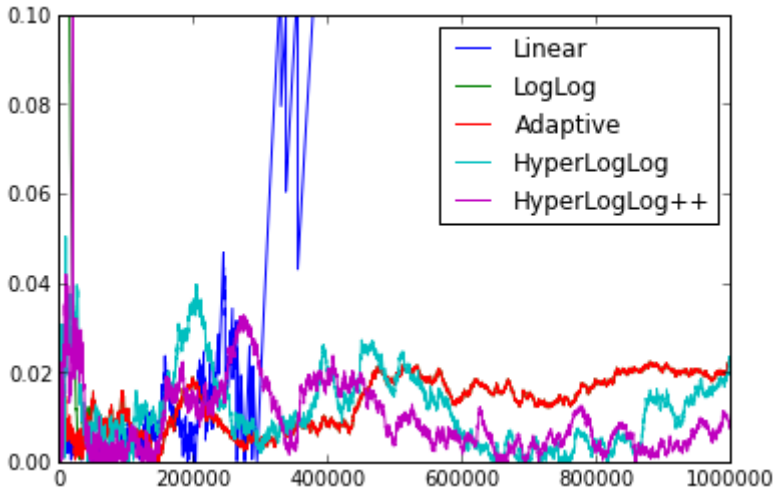
## 误差比较

为了更清楚对比五种算法的误差情况，下面给出五种算法的误差曲线叠加图，仍然是采用三个分桶数。

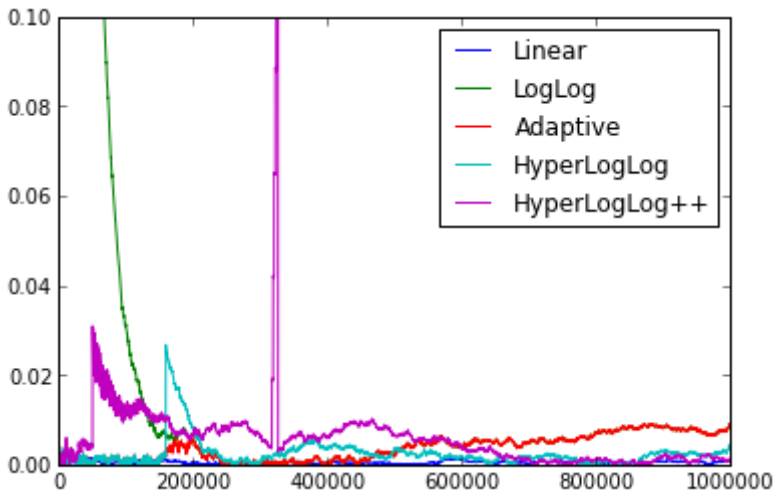
误差对比（p=10）



误差对比（p=12）




误差对比（p=16）




# 实践建议

下面根据实验结果从个人角度给出一些基数估计算法的实践性建议，当然只代表个人意见，不同人对实验结果可能有不同解读。

- Linear Counting和LogLog Counting由于分别在基数较大和基数较小（阈值可解析分析，具体方法和公式请参考后文列出的相关论文）时存在严重的失效，因此不适合在实际中单独使用。一种例外是，如果对节省存储空间要求不强烈，不要求空间复杂度为常数（Linear Counting的空间复杂度为$O(n)$，其它算法均为$O(1)$），则在保证bitmap全满概率很小的条件下，Linear Counting的效果要优于其它算法。
- 总体来看，不论哪种算法，提高分桶数都可以降低偏差和方差，因此总体来看基数估计算法中分桶数的选择是最重要的一个权衡——在精度和存储空间间的权衡。
- 实际中，Adaptive Counting或HyperLogLog Counting都是不错的选择，前者偏差较小，后者对离群点容忍性更好，方差较小。
- Google的HyperLogLog Counting++算法属于实验性改进，缺乏严格的数学分析基础，通用性存疑，不宜在实际中贸然使用。

# 参考文献

[1] K.-Y. Whang, B. T. Vander-Zanden, and H. M. Taylor. A Linear-Time Probabilistic Counting Algorithm for Database Applications. ACM Transactions on Database Systems, 15(2):208-229, 1990.

[2] Marianne Durand and Philippe Flajolet. LogLog counting of large cardinalities. In ESA03, volume 2832 of LNCS, pages 605b 617, 2003.

[3] Min Cai, Jianping Pan, Yu K. Kwok, and Kai Hwang. Fast and accurate traffic matrix measurement using adaptive cardinality counting. In MineNet b 05: Proceedings of the 2005 ACM SIGCOMM workshop on Mining network data, pages 205b 206, New York, NY, USA, 2005. ACM.

[4] P. Flajolet, E. Fusy, O. Gandouet, and F. Meunier. Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm. Disc. Math. and Theor. Comp. Sci., AH:127-146, 2007.

[5] HyperLogLog in Practice: Algorithmic Engineering of a State of The Art Cardinality Estimation Algorithm.

[6] Appendix to HyperLogLog in Practice: Algorithmic Engineering of a State of the Art Cardinality Estimation Algorithm.

---